Arbiter: A Domain-Specific Language for Ethical Machine Learning

Author 1 - Computer Science and **Author 2 - Computer Science and Philosophy**

Abstract

The widespread deployment of machine learning models in high-stakes decision making scenarios requires a code of ethics for machine learning practitioners. We identify four of the primary components required for the ethical practice of machine learning: transparency, fairness, accountability, and reproducibility. We introduce Arbiter, a domain-specific programming language for machine learning practitioners that is designed for ethical machine learning. Arbiter provides a notation for recording how machine learning models will be trained, and we show how this notation can encourage the four components of ethical machine learning.

1 Introduction

In this paper, we discuss what ethical machine learning is, demonstrate what a domain-specific programming language for ethical machine learning could look like, and demonstrate how that language will aid in the practice of ethical machine learning.

A domain-specific language (DSL) is "a computer programming language of limited expressiveness focused on a particular domain" (Fowler 2010). DSLs contrast with general-purpose languages, such as Python, which aim for universal applicability. When using a DSL, programmers can "[use] the language of the domain to state the problem and to articulate solution processes" (Felleisen et al. 2015), greatly increasing their productivity. The code they write can either look similar to the mathematical notation for the problem they are solving, such as in SPL (Werk, Ahnfelt-Rønne, and Larsen 2012), or look similar to a plain English description of the desired computation, such as in SQL (Date and Darwen 1997). The resemblance of SQL to plain English, for example, means that SQL imposes less of a cognitive burden on programmers than general-purpose languages do.

Machine learning models are often used in high-stakes decision making, such as in credit-scoring, housing, and hiring decisions. The outcomes of these decisions alter the life prospects of the decision-subjects, so it is important that the decisions are justified. For instance, a machine learning model used across America to determine the length of criminal sentences is biased against black people, which is an example of unethical machine learning (Kirchner et al. 2016). Machine learning models tend to "reproduce existing patterns of discrimination" (Barocas and Selbst 2016), so active interventions must be taken to ensure that models are fair. Fortunately, there are many such interventions, including pre-processing the input data, altering the training method, and post-processing the predictions made by the model (Bellamy et al. 2018; Hajian, Bonchi, and Castillo 2016). However, merely having a fair model is not enough: models must also be accountable to ensure that they are being used fairly (Binns 2018). Accountability is not just a property of the model, but also a property of the "training regime", the bundle of code and data used to develop the model. And reproducibility is required to ensure auditors can guarantee that a deployed model is equivalent to the models that were tested for fairness, and that the code they audit is the same code that was used to create a deployed model. Ethical machine learning is an underspecified term, but it definitely includes transparency, fairness, accountability, and reproducibility.

Using this definition, we will spend the rest of the paper arguing that DSLs can help practitioners practice ethical machine learning. In Section 2, we introduce some prior work in DSLs, particularly for machine learning. In Section 3, we define ethical machine learning more thoroughly, and highlight some of the key characteristics of the practice of ethical machine learning. In Section 4, we introduce Arbiter, demonstrating its syntax and the core features. In Sections 5 through 8, we demonstrate how Arbiter can help practitioners achieve four different components of the practice of ethical machine learning.

2 Previous Domain-Specific Languages for Machine Learning

Machine learning practitioners have designed and implemented many DSLs to aid the practice of machine learning. For instance, TensorFlow is a DSL for expressing the matrix multiplications that are required to implement a neural network. By expressing their computations within TensorFlow's domain-specific language, users can increase the performance and legibility of their code (Abadi et al. 2016). TensorFlow is an "internal" DSL. Internal DSLs are implemented as a library in another language, called the "host

language". For this reason, TensorFlow cannot make strong guarantees about the way the DSL will behave in all cases. The host language, Python, allows users to manipulate the programming language itself, and so TensorFlow code that seems to produce a fair model may only produce a fair model in certain environments. This uncertainty makes it impossible for machine learning models implemented in TensorFlow to be completely auditable and reproducible. On the other hand, an external DSL can make stronger guarantees about its outputs.

While there are no existing DSLs for ethical machine learning, there are libraries and toolkits for training fair models and evaluating the fairness of existing models. AI Fairness 360 is one prominent example (Bellamy et al. 2018). However, using AI Fairness 360 is an ad-hoc intervention, applied after the training of a model and potentially difficult for auditors to reproduce, because the fairness-testing code may not be stored in the same place as the code that produced the model. DSLs for ethical machine learning can overcome this problem by being integral to the training regime. Furthermore, libraries such as AI Fairness 360 can easily be misused by practitioners that do not understand how to use the library properly. AI Fairness 360 contains over 5000 lines of code (as counted by `cloc`), which is large enough that users may struggle to understand it fully, and requires that users convert their data into a format that AI Fairness 360 can use, creating the potential for further mistakes.

Outside of machine learning, DSLs have been used before to encourage or require best practices for programming. For instance, the LangSec movement is centered around the idea that security can be increased if "the acceptable input to a program [is] well-defined (i.e., via a grammar), as simple as possible (on the Chomsky scale of syntactic complexity), and fully validated before use" (Momot et al. 2016). In other words, LangSec requires that software engineers use a DSL for input validation. By explicitly limiting the set of valid inputs to a program, you can improve the security of that program. In a parallel to this idea, we will show that by explicitly limiting the set of expressible training regimes, we can encourage ethical machine learning practices.

3 Ethical Machine Learning

Ethical machine learning describes ethical, responsible, and thoughtful practices for the development and use of machine learning technology. Various ethicists, including Barocas (2016) and Binns (2017), have defended a number of principles as the core tenets of ethical machine learning; but they all agree that transparency, fairness, accountability, and reproducibility play important roles. Though this is not an exhaustive list of the tenets that constitute ethical machine learning, each of these principles is required for the responsible development and deployment of machine learning models.

Transparency refers to openness and understandability in the training of machine learning models. A training regime is transparent if auditors and the general public can understand the model. Transparency is required because, although black-box testing can be done on the resulting models, it is much easier to audit and understand a training regime than the resulting model (De Laat 2018). Ethical machine learning must include transparency, because auditors and the general public must understand how models are being trained in order to evaluate those models.

Fairness in machine learning has many definitions, but they are all measures of equality in the distribution of outcomes by models. The deployment of unfair machine learning systems "raises serious concerns because it risks limiting our ability to achieve the goals that we have set for ourselves and access the opportunities for which we are qualified" (Passi and Barocas 2019). However, defining fairness is difficult. AI Fairness 360 defines over 70 ways to measure fairness (Bellamy et al. 2018), and many of these metrics are contradictory, that is, an increase in one may necessitate a decrease in another (Kleinberg, Mullainathan, and Raghavan 2016). But measuring the fairness of models is still important, both for legal compliance (Barocas and Selbst 2016) and for ethical reasons (Binns 2018). Laws such as Title VII prohibit decision-making processes that have "disparate impact", that is, decision-making processing that give better outcomes to people in some protected classes than others (Barocas and Selbst 2016). And it is unethical to construct decision-making systems that you know will be unreasonably biased against people in marginalized groups.

Accountability in machine learning is defined by Binns (2017): a machine learning model is accountable if it "[provides] its decision-subjects with reasons and explanations" for the decisions it makes. Without the capacity or obligation to provide decision-subjects with explanations, practitioners can design training regimes that produce algorithmic decision-makers that are discriminatory, arbitrary, or otherwise unjust, and get away with it. Accountability compels the defense of design decisions, and in doing so compels ethical design.

Reproducibility in machine learning requires that practitioners have the ability to reproduce models independently. Olorisade (2017) argues that machine learning practitioners must work to increase reproducibility, so that other people will be able to recreate any produced models in order to test them. In order for other machine learning practitioners to recreate the model to test it for unfairness, the same training regime must produce the same model.

As the following sections will show, a domain-specific language can help machine learning practitioners uphold each of the tenets of ethical machine learning described above.

4 An Overview of the Language

The rest of this paper will examine Arbiter, a DSL [1] for specifying training regimes. Listing 1 is an example of a training regime, specified in Arbiter. All of the code in Listing 1 is explained in the following sections.

[1] An implementation of an arbiter-like language can be found at [ELIDED FOR PEER REVIEW]. This implementation proves that it would be feasible to implement Arbiter, but the point of this paper is to explain the ethical benefits of such a language existing, not to demonstrate an implementation.

```
FROM DATA 'credit_data.csv'
TRAIN A 'decision tree'
PREDICTING 'default'
WRITE MODEL TO 'credit_score.model'
PROTECTED CLASSES 'race', 'gender', 'age'
REQUIRED FAIRNESS 'disparate impact' < 1.1
EXPLANATION 'decision_reason'
```

Listing 1: Arbiter example.

This language is declarative, that is, users will "say what is required and let the system determine how to achieve it" (Van Roy and Haridi 2004), rather than "say[ing] how to do something" (Van Roy and Haridi 2004), as one must in imperative programming languages. The declarative style of programming is not strictly better than the imperative style: programming languages are tools that give us notations to express computation, and "a notation is never absolutely good ... but good only in relation to certain tasks." (Green 1989). However, as the rest of this paper will show, the declarative paradigm and the notation implemented by Arbiter is beneficial to the task of producing transparent, fair, accountable, and reproducible machine learning models. SQL, a domain-specific language for analysis and manipulation of tabular data, has benefitted similarly from its declarative style, which makes SQL both easier to read and faster to write than equivalent imperative code.

5 Improving Transparency

It is important that the general public as well as auditors can understand the training regime that was used to develop a model; otherwise they may not be able to identify flaws or bias being introduced during the training of the model. Training regimes specified in Arbiter are understandable by people who are not machine learning practitioners. For instance, lines 1 through 4 of Listing 1 describes a fairly archetypal modeling task: given a file with comma-separated data (a "CSV file") containing information about historical creditor defaults, it will produce a model that can predict whether a person is likely to default on a loan. The resulting model can be employed to replace human decision-making about creditworthiness. This model will likely be biased and difficult to audit, but those concerns are handled in lines 5 through 7, which will be discussed in the sections to come.

Arbiter is more transparent than imperative programming languages are. Arbiter only needs to express the essential aspects of the machine learning model training process, while a general-purpose language must concern itself with incidental aspects of the training process, such as reading the data file and converting it into the right data format. For example, the Python code below, which is equivalent to lines 1-3 of Listing 1 is much more difficult to read. While omitted for economy of presentation, adding the code that tests for and improves fairness more than triples the length of the Python code, but adding an equivalent step to the training regime in Arbiter only requires two lines of code, nameline lines 5 and 6.

```
with open('credit_data.csv') as credit_data:
    file_content = [line for line in csv.
    reader(credit_data)]
labels = [int(line[-1]) for line in
    file_content]
features = [[float(x) for x in line[0:-1]]
    for line in file_content]

from sklearn.model_selection import
    train_test_split
features_train, features_test, labels_train,
     labels_test = \
    train_test_split(features, labels)

from sklearn.tree import
    DecisionTreeClassifier
tree = DecisionTreeClassifier().fit(
    features_train, labels_train)
predictions = tree.predict(features_test)
print(f'Accuracy: {sum(predictions ==
    labels_test) / len(features_test)}')
```

Listing 2: Python code equivalent to Listing 1.

Arbiter code can ensure a higher level of transparency than libraries in general-purpose programming languages. By having fewer possible expressible computations, understanding a program requires less effort. Furthermore, even the best-designed library API can be used incorrectly, inelegantly, or incomprehensibly by code that calls the library; on the other hand, domain-specific languages can enforce consistent style and structure. Consistent style and structure, in turn, allow auditors who are less familiar with the language to nonetheless evaluate the program's behavior.

6 Improving Fairness

It is illegal and unethical to deploy models that are biased against people in marginalized classes. Arbiter allows users to make guarantees about the level of bias in any produced models. Perfect fairness of a model is impossible, but "surely some are fairer than others" (Grant 2019), and Arbiter encourages practitioners to use the fairer models. Lines 5 and 6 of Listing 1 specifies that columns of data named "race", "gender", and "age" represent marginalized classes, and that the resulting model must have disparate impact less than 1.1 for each of those classes. Disparate impact is a measure of unfair allocation of benefit across marginalized classes, calculated by taking the ratio of positive outcomes for the privileged class, and dividing by the ratio of positive outcomes for the marginalized class. A disparate impact of 1 represents benefit being perfectly evenly distributed, and numbers larger than 1 represent more unfair distributions. While this debiasing step will have to be optional, as not all datasets contain data about marginalized classes, the language could mandate that the `PROTECTED CLASSES` and `REQUIRED FAIRNESS` directives are included anyway, so that omitting a fairness requirement for your model is glaringly obvious to any auditor of the code, regardless of their familiarity with machine learning. If the absense of code to check for fairness is explicit, auditors will be less likely to overlook it than if it is merely not present.

Arbiter can use existing toolkits, such as AI Fairness 360, at multiple stages in the training process, in order to reduce bias. Practitioners will not need to know how to use these

(often complex) tools, because they can simply specify the level of fairness that they want to guarantee, and let Arbiter figure out how to accomplish that. Because Arbiter is a declarative language, practitioners only have to know the desired result, not the intermediate steps required to achieve that result. Using libraries to increase the fairness of models often requires reading documentation, converting your data to other data formats, and much additional work; many practitioners likely make mistakes in using these libraries and do not increase fairness as much as they could. Arbiter abstracts away this work, allowing one group of developers to write the code that interfaces with libraries that improve fairness to save all other developers from having to duplicate that work.

7 Improving Accountability

Accountability is a necessary piece of ethical machine learning. To be accountable, a model must "provide its decision-subjects with reasons and explanations" (Binns 2017) for the decisions it makes. Arbiter can ensure that models are explainable: because Arbiter holds the responsibility for training the model, the language can ensure that the model is trained in a way that includes explanations. For example, it could use TED (Hind et al. 2019), allowing users to specify a column of the input dataset that contains the explanation for the decisions made in the input dataset, as shown in line 7 of Listing 1. Then, the model produced can return not only a prediction but also an explanation of the reason for that prediction. Importantly, machine learning practitioners will not have to implement TED themselves, or know how it works – they will merely have to specify in their training regime that they want explanations for each prediction, and it will happen.

Furthermore, Arbiter could automatically produce tools for exploring and explaining the models that it generates. For instance, a tool that allows practitioners to apply the model to various data points to see what decision is made, and what explanation for the decision is given. Additional tooling that supports transparency can be produced automatically, without requiring any time or effort from the practitioner training the model, similarly to how the team behind TensorFlow created TensorBoard, a tool that automatically visualizes neural networks being trained in TensorFlow. Having tools for accountability readily available will increase the average accountability of models being created.

8 Improving Reproducibility

Training machine learning models is typically a non-deterministic process involving stochastic methods. These stochastic methods "seed" a random number generator with the current time, preventing future auditors from replicating the exact training process. But merely choosing a static "seed" value is not enough to guarantee reproducibility, as there are many other ways that software can be non-deterministic (Maste 2017), and the compilers or interpreters may contain vulnerabilities (Thompson and others 1984). So, auditors re-running a training regime will almost certainly end up with a different model. And once the two models are not exactly byte-for-byte identical, it becomes very difficult to identify the difference in behavior between the two (Perry, Schoen, and Steiner 2014).

However, a language like Arbiter can guarantee reproducibility. For example, by using a hash of the data in the training specification as the seed value, Arbiter can guarantee that every execution of the same training specification will be using the same random seed. Arbiter can be implemented to produce the same model for any input training regime, unlike Python libraries, which cannot guarantee reproducibility. Even if a Python library is itself reproducible, users may misuse it or write non-reproducible code that calls into it, resulting in non-deterministic training regimes. Training must be reproducible, and this can only be accomplished through a DSL. On the other hand, Arbiter has full control over how the model the user has declared will be trained, so it can ensure that it uses only reproducible libraries, and that those libraries are composed in a way that remains fully reproducible. Furthermore, in order to guarantee reproducibility in practice, many different inputs to Arbiter can be tested, and reproducibility can be demonstrated for each of those inputs. This is unlikely to happen in standard software engineering, where time constraints and fiscal limitations often prevent practitioners from testing their code for reproducibility.

9 Conclusion

We presented Arbiter, a DSL that can aid in the practice of ethical machine learning. Arbiter improves transparency, mandates fairness, enables accountability, and guarantees reproducibility in the machine learning training process. While Arbiter is not a silver bullet, it lays the groundwork for a programming-language-based approach to ethical machine learning. And, by establishing some desirable features of a DSL for ethical machine learning, we have provided a basis for further work in DSL design for ethical programming in general.

References

Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 265–283.

Barocas, S., and Selbst, A. D. 2016. Big data's disparate impact. *Calif. L. Rev.* 104:671.

Bellamy, R. K.; Dey, K.; Hind, M.; Hoffman, S. C.; Houde, S.; Kannan, K.; Lohia, P.; Martino, J.; Mehta, S.; Mojsilovic, A.; et al. 2018. Ai fairness 360: An extensible toolkit for detecting, understanding, and mitigating unwanted algorithmic bias. *arXiv preprint arXiv:1810.01943*.

Binns, R. 2017. Fairness in machine learning: Lessons from political philosophy. *arXiv preprint arXiv:1712.03586*.

Binns, R. 2018. Algorithmic accountability and public reason. *Philosophy & technology* 31(4):543–556.

Date, C., and Darwen, H. 1997. A guide to the sql standard: a user's guide to the standard database language.

De Laat, P. B. 2018. Algorithmic decision-making based on machine learning from big data: Can transparency restore accountability? *Philosophy & technology* 31(4):525–541.

Felleisen, M.; Findler, R. B.; Flatt, M.; Krishnamurthi, S.; Barzilay, E.; McCarthy, J.; and Tobin-Hochstadt, S. 2015. The racket manifesto. In *1st Summit on Advances in Programming Languages (SNAPL 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Fowler, M. 2010. *Domain-specific languages*. Pearson Education.

Grant, C. 2019. Is it impossible to be fair?

Green, T. R. 1989. Cognitive dimensions of notations. *People and computers V* 443–460.

Hajian, S.; Bonchi, F.; and Castillo, C. 2016. Algorithmic bias: From discrimination discovery to fairness-aware data mining. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2125–2126. ACM.

Hind, M.; Wei, D.; Campbell, M.; Codella, N. C.; Dhurandhar, A.; Mojsilović, A.; Natesan Ramamurthy, K.; and Varshney, K. R. 2019. Ted: Teaching ai to explain its decisions. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, 123–129. ACM.

Kirchner, L.; Angwin, J.; Larson, J.; and Mattu, S. 2016. Machine bias: There's software used across the country to predict future criminals, and it's biased against blacks. *Pro Publica: New York, NY, USA*.

Kleinberg, J. M.; Mullainathan, S.; and Raghavan, M. 2016. Inherent trade-offs in the fair determination of risk scores. *CoRR* abs/1609.05807.

Maste, E. 2017. Reproducible builds in freebsd.

Momot, F.; Bratus, S.; Hallberg, S. M.; and Patterson, M. L. 2016. The seven turrets of babel: A taxonomy of langsec errors and how to expunge them. In *2016 IEEE Cybersecurity Development (SecDev)*, 45–52. IEEE.

Olorisade, B. K.; Brereton, P.; and Andras, P. 2017. Reproducibility in machine learning-based studies: An example of text mining.

Passi, S., and Barocas, S. 2019. Problem formulation and fairness. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, 39–48. ACM.

Perry, M.; Schoen, S.; and Steiner, H. 2014. Reproducible builds. moving beyond single points of failure for software distribution. In *Chaos Communication Congress*.

Thompson, K., et al. 1984. Reflections on trusting trust. *Commun. ACM* 27(8):761–763.

Van Roy, P., and Haridi, S. 2004. Concepts, techniques, and models of computer programming. *The MIT Press, 2004. ISBN: 0262220695. 930pp.* 19:254 – 256.

Werk, M. F.; Ahnfelt-Rønne, J.; and Larsen, K. F. 2012. An embedded dsl for stochastic processes. In *FHPC@ ICFP*, 93–102.